One-Pass Algorithms for MAP Inference of Nonsymmetric Determinantal Point Processes

Aravind Reddy [1] **Ryan A. Rossi** [2] **Zhao Song** [2] **Anup Rao** [2] **Tung Mai** [2] **Nedim Lipka** [2] **Gang Wu** [2] **Eunyee Koh** [2] **Nesreen K. Ahmed** [3]

Abstract

In this paper, we initiate the study of one-pass algorithms for solving the maximum-a-posteriori (MAP) inference problem for Non-symmetric Determinantal Point Processes (NDPPs). In particular, we formulate streaming and online versions of the problem and provide one-pass algorithms for solving these problems. In our streaming setting, data points arrive in an arbitrary order and the algorithms are constrained to use a single-pass over the data as well as sub-linear memory, and only need to output a valid solution at the end of the stream. Our online setting has an additional requirement of maintaining a valid solution at any point in time. We design new one-pass algorithms for these problems and show that they perform comparably to (or even better than) the offline greedy algorithm while using substantially lower memory.

1. Introduction

Determinantal Point Processes (DPPs) were first introduced in the context of quantum mechanics (Macchi, 1975) and have subsequently been extensively studied with applications in several areas of pure and applied mathematics like graph theory, combinatorics, random matrix theory (Hough et al., 2006; Borodin, 2009), and randomized numerical linear algebra (Derezinski & Mahoney, 2021). Discrete DPPs have gained widespread adoption in machine learning following the seminal work of (Kulesza & Taskar, 2012) and have also seen a recent explosion of interest in the machine learning community. For instance, some of the very recent uses of DPPs include automation of deep neural network design (Nguyen et al., 2021), deep generative models (Chen & Ahmed, 2021), document and video summarization (Perez-Beltrachini & Lapata, 2021), image processing (Launay et al., 2021), and learning in games (Perez-Nieves et al., 2021).

A DPP is a probability distribution over subsets of items and is characterized by some kernel matrix such that the probability of sampling any particular subset is proportional to the determinant of the submatrix corresponding to that subset in the kernel. Until very recently, most prior work on DPPs focused on the setting where the kernel matrix is symmetric. Due to this constraint, DPPs can only model negative correlations between items. Recent work has shown that allowing the kernel matrix to be nonsymmetric can greatly increase the expressive power of DPPs and allows them to model *compatible* sets of items (Gartrell et al., 2019; Brunel, 2018). To differentiate this line of work from prior literature on symmetric DPPs, the term Nonsymmetric DPPs (NDPPs) has often been used. Modeling positive correlations can be useful in many practical scenarios. For instance, an E-commerce company trying to build a product recommendation system would want the system to increase the probability of suggesting a router if a customer adds a modem to a shopping cart.

State-of-the-art algorithms for MAP inference of NDPPs (Gartrell et al., 2021; Anari & Vuong, 2022) require storing the full data in memory and take multiple passes over the complete dataset. Therefore, these algorithms take too much memory to be useful for large scale data, where the size of the entire dataset can be much larger than the random-access memory available. These algorithms are also not practical in settings where data is generated on the fly, for example, in E-commerce applications where new items are added to the store over time.

Technical contributions.

- We give the first problem formulations for streaming and online versions of MAP Inference of low-rank NDPPs. Our formulations provide a good structure on how to store NDPP models which are so large that they cannot fit by themselves in the memory (RAM) of any single machine (this is an extremely important

[1]Northwestern University [2]Adobe Research [3]Intel Labs. Correspondence to: Aravind Reddy <aravind.reddy@cs.northwestern.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

practical problem due to the massive scale of industry datasets) i.e. store the $\mathbf{C}$ matrix separately and all the $\mathbf{v}_i$ and $\mathbf{b}_i$ as $(\mathbf{v}_i; \mathbf{b}_i)$ pairs (the straight-forward way to store the data would be to store $\mathbf{V}; \mathbf{C}; \mathbf{B}$ separately (as in the open-source code provided by Gartrell et al. (2021)).

- In Section 4, we provide our first streaming algorithm, PARTITION GREEDY, which is a streaming version of the standard greedy algorithm for submodular maximization(Nemhauser et al., 1978), and provide bounds for the approximation ratio, space used, and time taken.
- In Section 5, we provide ONLINE-LSS and ONLINE-2-NEIGHBOUR, our online algorithms based on local search with a stash, which are generalizations of the online local search algorithm for MAP Inference of symmetric DPPs by (Bhaskara et al., 2020), and provide bounds for the space used and time taken.
- In Section 6, we provide a hard instance for one-pass sublinear-space MAP Inference of NDPPs on which all of our algorithms fail to output solutions with a bounded approximation ratio. This illustrates that it might even be impossible to prove approximation factor guarantees for our algorithms without additional strong assumptions. The hard instance uses properties of NDPPs that differ from symmetric DPPs illustrating some of the divergence between them. We also provide some additional comparison between MAP Inference of nonsymmetric DPPs and symmetric DPPs in this section.
- In Section 7, we evaluate our proposed online NDPP MAP Inference algorithms on several datasets and show that they show that they perform comparably to (or even better than) the offline greedy algorithm which takes multiple passes over the data and also uses substantially more memory (linear in number of items).

2. Related Work

Nonsymmetric DPPs. A special subset of NDPPs called signed DPPs were the first class of NDPPs to be studied (Brunel et al., 2017). Gartrell et al. (2019) studied a more general class of NDPPs and provided learning and MAP Inference algorithms, and also showed that NDPPs have additional expressiveness over symmetric DPPs and can better model certain problems. This was improved by Gartrell et al. (2021) in which they provided a new decomposition which enabled linear time learning and MAP Inference for NDPPs. More recently, Anari & Vuong (2022) proposed the first algorithm with a $k^{O(k)}$ approximation factor for MAP Inference on NDPPs where k is the number of items to be selected.

DPP MAP Inference. MAP Inference in DPPs is a very well studied NP-hard problem (Ko et al., 1995) with numerous applications in machine learning (Gillenwater et al., 2012; Han et al., 2017; Chen et al., 2018; Han & Gillenwater, 2020). Since matrix log-determinant is a submodular function, several offline algorithms for submodular function maximization (Krause & Golovin, 2014) have been applied to the problem. For instance, Civril & Magdon-Ismail (2009) showed that the standard greedy algorithm of Nemhauser et al. (1978) provides an $O(k!)$ factor approximation. Nevertheless, even in the case of symmetric DPPs, the study of streaming and online algorithms is in a nascent stage. In particular, (Indyk et al., 2019; 2020; Mahabadi et al., 2020) provided streaming algorithms for MAP inference of DPPs and (Bhaskara et al., 2020) were the first to propose online algorithms for MAP inference of DPPs. Also, (Liu et al., 2021) designed the first streaming algorithms for the maximum induced cardinality objective proposed by (Gillenwater et al., 2018). However, there has not yet been any work other than ours which has focused on either streaming or online algorithms for NDPPs.

Streaming and Online Algorithms. Streaming (Alon et al., 1999; Muthukrishnan, 2005) and online (Karp et al., 1990; Karp, 1992; Borodin & El-Yaniv, 2005) algorithms have been extensively studied in theoretical computer science. In particular, they have also seen many applications in machine learning such as reinforcement learning (Shrivastava et al., 2021), projected gradient descent (Xu et al., 2021), training over-parameterized neural networks (Song et al., 2021a;b), and solving linear programs (Song & Yu, 2021).

3. Preliminaries

3.1. Notation

Throughout the paper, we use uppercase bold letters ($\mathbf{A}$) to denote matrices and lowercase bold letters ($\mathbf{a}$) to denote vectors. Letters in normal font (a) will be used for scalars. For any positive integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. A matrix $\mathbf{M}$ is said to be skew-symmetric if $\mathbf{M} = -\mathbf{M}^\top$ where $\top$ is used to represent matrix transposition.

3.2. Background on DPPs

A DPP is a probability distribution on all subsets of $[n]$ characterized by a matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$. The probability of sampling any subset $S \subseteq [n]$ i.e. $\Pr[S] \propto \det(\mathbf{L}_S)$ where $\mathbf{L}_S$ is the submatrix of $\mathbf{L}$ obtained by keeping only the rows and columns corresponding to indices in S. The normalization constant for this distribution can be computed efficiently since we know that $\sum_{S \subseteq [n]} \det(\mathbf{L}_S) = \det(\mathbf{L} + \mathbf{I}_n)$ (Kulesza & Taskar, 2012, Theorem 2.1). Therefore,

Table 1. Summary of our MAP inference algorithms for NDPPs. We omit O for simplicity. All algorithms use only a single-pass over the data.

Inference Problem	Algorithm	Update Time	Total Time	Space
Streaming	STREAM-PARTITION (Alg. 1)	N/A	$T_{det}(k,d) \cdot n$	$k^2 + d^2$
Online	ONLINE-LSS (Alg. 2)	$T_{det}(k,d) \cdot k \log^2(\,)$	$T_{det}(k,d) \cdot (nk + k \log^2(\,))$	$k^2 + d^2 + d \log(\,)$
	ONLINE 2-NEIGH (Alg. 3)	$T_{det}(k,d) \cdot k^2 \log^3(\,)$	$T_{det}(k,d) \cdot (nk^2 + k^2 \log^3(\,))$	$k^2 + d^2 + d \log(\,)$
	ONLINE-GREEDY (Alg. 4)	$T_{det}(k,d) \cdot k$	$T_{det}(k,d) \cdot nk$	$k^2 + d^2$

$\Pr[S] = \frac{\det(L_S)}{\det(L + I_n)}$. For the DPP corresponding to L to be a valid probability distribution, we need $\det(L_S) \geq 0$ for all $S \subseteq [n]$ since $\Pr[S] \geq 0$ for all $S \subseteq [n]$. Matrices which satisfy this property are known as P_0-matrices (Fiedler & Pták, 1966). For any symmetric matrix L, $\det(L_S) \geq 0$ for all $S \subseteq [n]$ if and only if L is positive semi-definite (PSD) i.e. $x^T L x \geq 0$ for all $x \in \mathbb{R}^n$. Therefore, all symmetric matrices which correspond to valid DPPs are PSD. But there are P_0-matrices which are not necessarily symmetric (or even positive semi-definite). For example, $L = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ is a nonsymmetric P_0 matrix.

Any matrix L can be uniquely written as the sum of a symmetric and skew-symmetric matrix: $L = (L + L^\top)/2 + (L - L^\top)/2$. For the DPP characterized by L, the symmetric part of the decomposition can be thought of as encoding negative correlations between items and the skew-symmetric part as encoding positive correlations. Gartrell et al. (2019) proposed a decomposition which covers the set of all nonsymmetric PSD matrices (a subset of P_0 matrices) which allowed them to provide a cubic time algorithm (in the ground set size) for NDPP learning. This decomposition is $L = V^\top V + (BC^\top - CB^\top)$. Gartrell et al. (2021) provided more efficient (linear time) algorithms for learning and MAP inference using a new decomposition $L = V^\top V + B^\top CB$. Although both these decompositions only cover a subset of P_0 matrices, it turns out that they are quite useful for modeling real world instances and provide improved results when compared to (symmetric) DPPs.

For the decomposition $L = V^\top V + B^\top CB$, we have $V, B \in \mathbb{R}^{d \times n}$, $C \in \mathbb{R}^{d \times d}$ and C is skew-symmetric. Here we can think of the items having having a latent low-dimensional representation (v_i, b_i) where $v_i, b_i \in \mathbb{R}^d$. Intuitively, a low-dimensional representation (when compared to n) is sufficient for representing items because any particular item only interacts with a small number of other items in real-world datasets, as evidenced by the fact that the maximum basket size encountered in real-world data is much smaller than n.

3.3. Background on Streaming and Online Algorithms

The main difference between streaming and online algorithms are their parameters of interest. In streaming algorithms (Muthukrishnan, 2005), the main focus is on the solution quality at the end of the stream and memory used by the algorithm throughout the stream. Instead, in online algorithms (Karp et al., 1990), the main focus is on the solution quality at every time step and update time after seeing a new input. In the streaming and online models we will define, the online setting is a more restrictive version of the streaming setting. Therefore, for us, any algorithms which are valid online algorithms are also valid streaming algorithms. However, not all streaming algorithms are online algorithms. For instance, our main streaming algorithm (Algorithm 1) is not a valid online algorithm.

4. Streaming MAP Inference

In this section, we formulate the streaming MAP inference problem for NDPPs and design an algorithm for this problem with guarantees on the solution quality, space, and time.

4.1. Streaming MAP Inference Problem

We study the MAP Inference problem in low-rank NDPPs in the streaming setting where we see columns of a $2d \times n$ matrix in order (column-arrival model). Given some fixed skew-symmetric matrix $C \in \mathbb{R}^{d \times d}$, consider a stream of $2d$-dimensional vectors (which can be viewed as pairs of d-dimensional vectors) arriving in order:

$$(v_1, b_2), (v_2, b_2), \ldots, (v_n, b_n) \text{ where } v_t, b_t \in \mathbb{R}^d, \forall t \in [n]$$

The main goal in the streaming setting is to output the maximum likelihood subset $S \subseteq [n]$ of cardinality k at the end of the stream assuming that S is drawn from the NDPP characterized by $L = V^\top V + B^\top CB$ i.e.

$$\begin{aligned} S &= \operatorname*{argmax}_{S \subseteq [n], |S| = k} \det(L_S) \qquad (1) \\ &= \operatorname*{argmax}_{S \subseteq [n], |S| = k} \det(V_S^\top V_S + B_S^\top C B_S) \end{aligned}$$

For any $S \subseteq [n]$, $V_S \in \mathbb{R}^{d \times |S|}$ is the matrix whose each col-

Algorithm 1 Streaming Partition Greedy MAP Inference for low-rank NDPPs

```
Input: Length of the stream n and a stream of data points {(v_1, b_1), (v_2, b_2), ..., (v_n, b_n)}
Output: A solution set S of cardinality k at the end of the stream.
S_0 ← ∅, s_0 ← ∅
while new data (v_t, b_t) arrives in stream at time t do
    i ← ⌈tk/n⌉
    if f(S_{i-1} ∪ {t}) > f(S_{i-1} ∪ {s_i}) then
        s_i ← t
    if t is a multiple of n/k then
        S_i ← S_{i-1} ∪ s_i
        s_i ← ∅
return S_k
```

umn corresponds to v_i; $i \in S$g. Similarly, $B_S \in \mathbb{R}^{d \times |S|}$ is the matrix whose columns correspond to b_i; $i \in S$g. In the case of symmetric DPPs, this maximization problem in the non-streaming setting corresponds to MAP Inference in cardinality constrained DPPs, also known as k-DPPs (Kulesza & Taskar, 2011).

Definition 1. Given three matrices $V \in \mathbb{R}^{d \times k}$; $B \in \mathbb{R}^{d \times k}$ and $C \in \mathbb{R}^{d \times d}$, let $T_{det}(k; d)$ denote the running time of computing $\det(V^{\top}V + B^{\top}CB)$. We can take $T_{det}(k; d)$ being $O(kd^2)$ as a crude estimate.

Note that $T_{det}(k; d) = 2T_{mat}(d; k; d) + T_{mat}(d; d; k) + T_{mat}(k; k; k)$ where $T_{mat}(a; b; c)$ is the time required to multiply two matrices of dimensions $a \times b$ and $b \times c$. We have the last $T_{mat}(k; k; k)$ term because computing the determinant of a $k \times k$ matrix can be done (essentially) in the same time as computing the product of two matrices of dimension $k \times k$ (Aho et al., 1974, Theorem 6.6).

We will now describe a streaming algorithm for MAP inference in NDPPs, which we call the "Streaming Partition Greedy" algorithm.

4.2. Streaming Partition Greedy

Outline of Algorithm 1 : Our algorithm picks the first element of the solution greedily from the first set of $\frac{n}{k}$ elements, the second element from the next sequence of $\frac{n}{k}$ elements and so on. As described in Algorithm 1, let us use $S_0, S_1, \ldots, S_k$ to denote the solution sets maintained by the algorithm, where S_i represents the solution set of size i. In particular, we have that $S_i = S_{i-1} \cup \{s_i\}$ where $s_i = \arg\max_{j \in P_i} f(S \cup \{j\})$ and P_i denotes the i'th partition of the data i.e. $P_i := \{\frac{(i-1)n}{k} + 1, \frac{(i-1)n}{k} + 2, \ldots, \frac{in}{k}\}$.

$$\underbrace{(v_1, b_1), (v_2, b_2), \ldots, (v_{n/k}, b_{n/k})}_{P_1}, \ldots, \underbrace{(v_{(n/k)+1}, b_{(n/k)+1}), \ldots, (v_n, b_n)}_{P_k} \tag{2}$$

Theorem 2. For a random-order arrival stream, if S is the solution output by Algorithm 1 at the end of the stream and $\sigma_{min} > 1$ where σ_{min} and σ_{max} denote the smallest and largest singular values of L_S among all $S \subseteq [n]$ and $|S| \le 2k$, then

$$\frac{\mathbb{E}[\log\det(L_S)]}{\log(OPT)} \ge \left(1 - \frac{1}{(1-\frac{1}{e})(2\log\sigma_{max} - \log\sigma_{min})}\right)$$

where $L_S = V_S^{\top}V_S + B_S^{\top}CB_S$ and $OPT = \max_{R \subseteq [n];\, |R| = k} \det(V_R^{\top}V_R + B_R^{\top}CB_R)$.

We will first give a high-level proof sketch for this theorem and defer the full proof to Appendix A.

Proof sketch. For a random-order arrival stream, the distribution of any set of consecutive n/k elements is the same as the distribution of n/k elements picked uniformly at random (without replacement) from $[n]$. If the objective function f is submodular, then this algorithm has an approximation guarantee of $(1 - 2/e)$ by (Mirzasoleiman et al., 2015). But neither $\det(L_S)$ nor $\log\det(L_S)$ are submodular. Instead, (Gartrell et al., 2021) [Equation 45] showed that $\log\det(L_S)$ is "close" to submodular when $\sigma_{min} > 1$ where this closeness is measured using a parameter known as "submodularity ratio" (Bian et al., 2017). Using this parameter, we can prove a guarantee for our algorithm.

Theorem 3. For any length-n stream $(v_1, b_1), \ldots, (v_n, b_n)$ where $(v_t, b_t) \in \mathbb{R}^d \times \mathbb{R}^d\ \forall t \in [n]$, the space used is $O(k^2 + d^2)$ and the total time taken is $O(n \cdot T_{det}(k; d))$ where $T_{det}(k; d)$ is the time taken to compute $f(S) = \det(V_S^{\top}V + B_S^{\top}CB)$ for $|S| = k$.

Proof. For any particular data-point (v_t, b_t), Algorithm 1 needs to compute $f(S_{i-1} \cup \{t\})$, where $f(S) = \det(V_S^{\top}V + B_S^{\top}CB_S)$ and $S = S_{i-1} \cup \{t\}$. This takes at most $O(k^2 + d^2)$ space and $T_{det}(k; d)$ time. The algorithm also needs to store S_{i-1}; s_i and $f(S_{i-1} \cup \{s_i\})$ but all of these are dominated by $O(k^2 + d^2)$ space needed to compute the determinant. All the other comparison and update steps are also much faster and so the total time is $O(n \cdot T_{det}(k; d))$

Algorithm 2 ONLINE-LSS: Online MAP Inference for low-rank NDPPs with Stash.

```
1:  Input: A stream of data points {(v_1, b_1), ..., (v_n, b_n)}, and a constant α ≥ 1
2:  Output: A solution set S of cardinality k at the end of the stream.
3:  S, T ← ∅
4:  while new data point (v_t, b_t) arrives in stream at time t do
5:      if |S| < k and f(S ∪ {t}) ≠ 0 then
6:          S ← S ∪ {t}
7:      else
8:          i ← argmax_{j ∈ S} f(S ∪ {t} \ {j})
9:          if f(S ∪ {t} \ {i}) > α · f(S) then
10:             S ← S ∪ {t} \ {i}
11:             T ← T ∪ {i}
12:             while ∃ a ∈ S, b ∈ T : f(S ∪ {b} \ {a}) > α · f(S) do
13:                 S ← S ∪ {b} \ {a}
14:                 T ← T ∪ {a} \ {b}
15: return S
```

5. Online MAP Inference for NDPPs

We now consider the online MAP inference problem for NDPPs, which is natural in settings where data is generated on the fly. In addition to the constraints of the streaming setting (Section 4.1), our online setting requires us to maintain a valid solution at every time step. In this section, we provide two algorithms for solving this problem.

5.1. Online Local Search with a Stash

Outline of Algorithm 2: On a high-level, our algorithm is a generalization of the Online-LS algorithm for DPPs from (Bhaskara et al., 2020). At each time step $t \in [n]$ (after $t \ge k$), our algorithm maintains a candidate solution subset of indices S of cardinality k from the data seen so far i.e. $S \subseteq [t]$ s.t. $|S| = k$ in a streaming fashion. Additionally, it also maintains two matrices $V_S, B_S \in \mathbb{R}^{d \times |S|}$ where the columns of V_S are $\{v_i; i \in S\}$ and the columns of B_S are $\{b_i; i \in S\}$. Whenever the algorithm sees a new data point (v_t, b_t), it replaces an existing index from S with the newly arrived index if doing so increases $f(S)$ at-least by a factor of $\alpha \ge 1$ where α is a parameter to be chosen (we can think of α being 2 for understanding the algorithm). Instead of just deleting the index replaced from S, it is stored in an auxiliary set T called the "stash" (and also maintains corresponding matrices V_T, B_T), which the algorithm then uses to performs a local search over to find a locally optimal solution.

We now define a data-dependent parameter Δ which we will need to describe the time and space used by Algorithm 2.

Definition 4. Let the first non-zero value of $f(S)$ with $|S| = k$ that can be achieved in the stream without any swaps be val_{nz} i.e. till S reaches a size k, any index seen is added to S if $f(S)$ remains non-zero even after adding it. Let us define $\Delta := \frac{OPT_k}{val_{nz}}$ where $OPT_k = \max_{S \subseteq [n]; |S| = k} \det(V_S^\top V_S + B_S^\top C B_S)$.

Note that Δ is a data-dependent parameter which can potentially be unbounded. This happens in the hard-instance we will describe in Section 6. However, for practical datasets, Δ doesn't seem to be too bad (see the experiments section for a more detailed discussion).

Theorem 5. For any length n stream $(v_1, b_1), \ldots, (v_n, b_n)$ where $(v_t, b_t) \in \mathbb{R}^d \times \mathbb{R}^d \ \forall t \in [n]$, the worst case update time of Algorithm 2 is $O(T_{det}(k, d) \cdot k \log^2(\Delta))$ where $T_{det}(k, d)$ is the time taken to compute $f(S) = \det(V_S^\top V + B_S^\top C B)$ for $|S| = k$. Furthermore, the amortized update time is $O(T_{det}(k, d) \cdot (k + \frac{k \log^2(\Delta)}{n}))$ and the space used at any time step is at most $O(k^2 + d^2 + d \log(\Delta))$.

Proof. For every iteration of the while loop in line 4. It takes at most $T_{det}(k, d)$ time for checking the first if condition (lines 5-6). The $\arg\max_{j \in S} f(S \cup \{t\} \setminus \{j\})$ step takes at most $k \cdot T_{det}(k, d)$ time. The while loop in line 12 takes time at most $|S| \cdot |T| \cdot T_{det}(k, d)$ for every instance of an increase in $f(S)$. Note that $f(S)$ can increase at most $\log(\Delta)$ times since the value of $f(S)$ cannot exceed OPT_k. Therefore, the update time of Algorithm 2 is at most $T_{det}(k, d) + k \cdot T_{det}(k, d) + \log(\Delta) \cdot (|S| \cdot |T| \cdot T_{det}(k, d)) \le T_{det}(k, d) \cdot (k + 1 + k \log^2(\Delta))$ since $|S| \le k$ and $|T| \le \log(\Delta)$. Notice that the cardinality of T can increase by 1 only when the value of $f(S)$ increases at least by a factor of α and so $|T| \le \log(\Delta)$.

During any time step t, the algorithm needs to store the indices in S, T and the corresponding matrices V_S, B_S, V_T, B_T. Since $|S| \le k, |T| \le \log(\Delta)$ and it takes d words to store every $\mathbf{v}_i$ and b_i, we need at most $(k + \log(\Delta)) \cdot 2d \le 2dk + 2d \log(\Delta)$ words to store all these in memory. The space needed to compute $\det(V_S^\top V_S + B_S^\top C B_S)$ is at most $O(k^2 + d^2)$. We compute all such determinants one after the other in our algorithm. So the algorithm only needs space for one such computation during it's run. Therefore, the space required by Algorithm 2 is $O(k^2 + d^2 + d \log(\Delta))$. $\square$

5.2. Online 2-neighborhood Local Search Algorithm with a Stash

Before we describe our algorithm, we will define a neighborhood of any solution, which will be useful for describing the local search part of our algorithm.

Definition 6 ($N_r(S; T)$). For any natural number $r \ge 1$ and any sets S, T we define the r-neighborhood of S with

Algorithm 3 ONLINE-2-NEIGHBOR: Local Search over 2-neighborhoods with Stash for Online NDPP MAP Inference.

1: **Input:** A stream of data points $\{(v_1, b_1), (v_2, b_2), \ldots, (v_n, b_n)\}$, and a constant $\epsilon \geq 1$
2: **Output:** A solution set S of cardinality k at the end of the stream.
3: $S, T \leftarrow \emptyset$
4: **while** new data (v_t, b_t) arrives in stream at time t **do**
5:   **if** $|S| < k$ and $f(S \cup \{t\}) \neq 0$ **then**
6:     $S \leftarrow S \cup \{t\}$
7:   **else**
8:     $\{i, j\} \leftarrow \arg\max_{a,b \in S} (f(S \cup \{t\} \setminus \{a\}), f(S \cup \{t, t-1\} \setminus \{a, b\}))$
9:     $f_{max} \leftarrow \max_{a,b \in S} (f(S \cup \{t\} \setminus \{a\}), f(S \cup \{t, t-1\} \setminus \{a, b\}))$
10:     **if** $f_{max} > \epsilon f(S)$ **then**
11:       **if** two items are chosen to be replaced **then**
12:         $S \leftarrow S \cup \{t, t-1\} \setminus \{i, j\}$
13:         $T \leftarrow T \cup \{i, j\}$
14:       **else**
15:         $S \leftarrow S \cup \{t\} \setminus \{i\}$
16:         $T \leftarrow T \cup \{i\}$
17:       **while** $\exists\, a, b \in S;\ c, d \in T : f(S \cup \{c, d\} \setminus \{a, b\}) > \epsilon f(S)$ **do**
18:         $S \leftarrow S \cup \{c, d\} \setminus \{a, b\}$
19:         $T \leftarrow T \cup \{a, b\} \setminus \{c, d\}$
20: **return** S

respect to T

$$N_r(S; T) := \{S' \subseteq S \cup T \mid |S'| = |S| \text{ and } |S' \setminus S| \leq r\}$$

Outline of Algorithm 3: Similar to Algorithm 2, our new algorithm also maintains two subsets of indices S and T, and corresponding data matrices $V_S, B_S; V_T, B_T$. Whenever the algorithm sees a new data-point (v_t, b_t), it checks if the solution quality $f(S)$ can be improved by a factor of ϵ by replacing any element in S with the newly seen data-point. Additionally, it also checks if the solution quality can be made better by including both the points (v_t, b_t) and the data-point (v_{t-1}, b_{t-1}). Further, the algorithm tries to improve the solution quality by performing a local search on $N_2(S; T)$ i.e. the neighborhood of the candidate solution S using the stash T by replacing at most two elements of S. There might be interactions captured by pairs of items which are much stronger than single items in NDPPs (see example 5 from (Anari & Vuong, 2022)).

Theorem 7. *For any length-n stream $(v_1, b_1), \ldots, (v_n, b_n)$ where $(v_t, b_t) \in \mathbb{R}^d \times \mathbb{R}^d\ \forall t \in [n]$, the worst case update time of Algorithm 3 is $O(T_{det}(k, d) \cdot k^2 \log^3(\kappa))$ where $T_{det}(k, d)$ is the time taken to compute $f(S) = \det(V_S^\top V + B_S^\top C B)$ for $|S| = k$. The amortized update time is $O\left(T_{det}(k, d) \cdot \left(k^2 + \frac{k^2 \log^3(\kappa)}{n}\right)\right)$ and the space used at any time step is at most $O(k^2 + d^2 + d \log(\kappa))$.*

Proof. It takes at most $T_{det}(k, d)$ time for lines 5-6 (same as in LSS). The $\arg\max_{a,b \in S} (f(S \cup \{t\} \setminus \{a\}), f(S \cup \{t, t-1\} \setminus \{a, b\}))$ step takes at most $k^2 \cdot T_{det}(k, d)$ time. The while loop in line 18 takes time at most $|S|^2 \cdot |T|^2 \cdot T_{det}(k, d)$ for every instance of an increase in $f(S)$. Similar to LSS, $f(S)$ can increase at most by a factor $\log_\epsilon(\kappa)$ since the value of $f(S)$ cannot exceed OPT_k. Therefore, the update time of Algorithm 3 is at most $T_{det}(k, d) + k^2 \cdot T_{det}(k, d) + \log_\epsilon(\kappa) \cdot |S|^2 \cdot |T|^2 \cdot T_{det}(k, d) \leq T_{det}(k, d) \cdot (k^2 + 1 + k^2 \log^3(\kappa))$ since $|S| \leq k$ and $|T| \leq \log(\kappa) \cdot k$.

Although Algorithm 3 executes more number of determinant computations than Algorithm 2, all of them are done sequentially and only the maximum value among all the previously computed values in any specific iteration needs to be stored in memory. Therefore, the space needed is (nearly) the same for both the algorithms.

6. Hard Instance for One-Pass MAP Inference of NDPPs

We will now give a high-level description of a hard instance for one-pass MAP inference of NDPPs with sub-linear memory (inspired by (Anari & Vuong, 2022, Example 5)) on which all of our algorithms output solutions with zero objective value whereas the optimal solution has non-zero value. Due to this, we believe it might be impossible to prove any bounded approximation factor guarantees for our algorithms without any strong additional assumptions.

Sketch of the hard instance. Suppose we have a total of $2d$ items consisting of *pairs* of complementary items like modem-router, printer-ink cartridge, pencil-eraser etc. Let us use $\{1, 1^c, 2, 2^c, \ldots, d, d^c\}$ to denote them. Any item i is independent of every item other than it's complement i^c. Individually, $\Pr[\{i\}] = \Pr[\{i^c\}] = 0$. And $\Pr[\{i, i^c\}] = x_i^2$ with $x_i > 0$ for all $i \in [d]$. Also, we have $\Pr[\{i, j\}] = 0$ for any $i \neq j$. Suppose any of our online algorithms are given the sequence $\{1, 2, 3, \ldots, d, r^c\}$ where $r \in [d]$ is some arbitrary item and the algorithm needs to pick 2 items i.e. $k = 2$. Then, $OPT > 0$ whereas all of our online algorithms (Online LSS, Online 2-neighbor, Online-Greedy) will fail to output a valid solution. We provide a more complete description with the instantiations of the matrices B, C, V in Appendix B.

Algorithm 4 ONLINE-GREEDY: Online Greedy MAP Inference for NDPPs

```
Input: A stream of data points {(v_1, b_1), (v_2, b_2), ..., (v_n, b_n)}
Output: A solution set S of cardinality k at the end of the stream.
S ← ∅
while new data (v_t, b_t) arrives in stream at time t do
    if |S| < k and f(S ∪ {t}) ≠ 0 then
        S ← S ∪ {t}
    else
        i ← argmax_{j ∈ S} f(S ∪ {t} \ {j})
        if f(S ∪ {t} \ {i}) > f(S) then
            S ← S ∪ {t} \ {i}
return S
```

Comparison between MAP Inference for NDPPs and symmetric DPPs. The hard instance described above necessarily uses the skew-symmetric component of the kernel matrix to form such *complementary* pairs and this illustrates some of the divergence between NDPPs and symmetric DPPs. NDPPs are significantly more general (and complex) than DPPs and unlike the case for DPPs where the objective function corresponds to a nice geometric notion i.e. volume, the objective function for MAP Inference on NDPPs doesn't have a corresponding clean notion. This is a core issue because of which it is unclear how the proofs of approximation factors for similar algorithms for DPPs would generalize to NDPPs (the proof techniques for DPPs from Bhaskara et al. (2020) for Online-LS heavily use the fact that the objective function is a volume and thus use properties of coresets developed for related geometric problems).

7. Experiments

We first learn all the matrices $B, C,$ and V by applying the learning algorithm of (Gartrell et al., 2021) on several real-world datasets. For example, some datasets consist of carts of items bought by Amazon customers. Then, we run our inference algorithms on the learned $B, C,$ and V. More details about the experiments and the datasets used can be found in Appendix C.

As a point of comparison, we also use the offline greedy algorithm from (Gartrell et al., 2021). This algorithm stores all data in memory and makes k passes over the dataset and in each round, picks the data point which gives the maximum marginal gain in solution value. Online-Greedy (Algorithm 4) is a simple online variant of the greedy algorithm which replaces a point in the current solution set with the observed point if doing so increases the objective.

First, we want to mention that the performance of our streaming algorithm (Algorithm 1) on the datasets we consider is (unfortunately) pretty bad (the objective function value is close to zero in most cases). This is because in real-world datasets, adjacent pairs of items have a very high likelihood of occurring together when compared to pairs of items far from each other. For example, white socks and grey socks might be adjacent to each other in numbering. And customers tend to buy both of them in a basket. Our streaming algorithm is forced to ignore most such pairs.

Results for a various datasets for our online algorithms are provided in Figure 1. Surprisingly, the solution quality of our online algorithms compare favorably with the offline greedy algorithm while using only a single-pass over the data, and a tiny fraction of memory. In most cases, Online-2-neighbor (Algorithm 3) performs better than Online-LSS (Algorithm 2) which in turn performs better than the online greedy algorithm (Algorithm 4). Strikingly, our online-2-neighbor algorithm performs even better than offline greedy in many cases.

We also perform several experiments comparing the number of determinant computations (as a system-independent proxy for time) and the number of swaps (as a measure of solution consistency) of all our online algorithms. Results for determinant computations (Figure 2) and swaps (Figure 3) can be found in Appendix D. We summarize the main findings here. The number of determinant computations of Online-LSS is comparable to that of Online Greedy but the number of swaps performed is significantly smaller. Online-2-neighbor is the most time-consuming but superior performing algorithm in terms of solution quality.

Our experimental results in Appendix D demonstrate that our online algorithms use substantially lower memory than any offline algorithms. Note that the main memory bottleneck for offline inference algorithms (Gartrell et al., 2021; Anari & Vuong, 2022) is the need to store the entire data-set in memory. We can consider other factors (like the memory needed for computing $\det(k, d)$ i.e. $O(k^2 + d^2)$ (which can

Figure 1. Solution quality i.e. objective function value as a function of the number of data points analyzed for all our online algorithms and also the offline greedy algorithm. All our online algorithms give comparable (or even better) performance to offline greedy using only a single pass and a small fraction of the memory.

be re-used every-time) to be essentially free because the regime of interest is n d k. The memory usage by our online algorithms is also primarily dominated by the size of the stash, which is upper bounded by the number of swaps for which we have plots in Appendix D.2. Similarly, the update times also depend only on the size of the stash (which are quite small and so we have very fast update times).

We also investigate the performance of our algorithms under the random stream paradigm, where we consider a random permutation of some of the datasets used earlier. Results for the solution quality (Figure 4), number of determinant computations and swaps (Figure 5) can be found in Appendix D.3. In this setting, we see that Online-LSS and Online-2-neighbor have nearly identical performance and are always better than Online-Greedy in terms of solution quality and number of swaps.

We study the effect of varying in Online-LSS (Algorithm 2) for various values of set sizes k in Appendices D.4 and D.5. We notice that, in general, the solution quality, number of determinant computations, and the number of swaps increase as decreases (Figure 6). We also see that as k increases, the solution value decreases across all values of (Figure 7. This is in accordance with our intuition that the probability of larger sets should be smaller.

8. Conclusion & Future Directions

In this paper, we formulate and study the streaming and online MAP inference problems for Nonsymmetric Determinantal Point Processes. To the best of our knowledge, this is the first work to study these problems in these practical settings. We design new one-pass algorithms for these problems, prove theoretical guarantees for them in terms of space required, time taken, and solution quality for our algorithms, and empirically show that they perform comparably or sometimes even better than state-of-the-art offline algorithms while using substantially lower memory.

As we have discussed in the experiments section, the empirical performance of our partition greedy algorithm is quite bad. The main reason we have chosen to include it in this paper is because it is the only one for which we have a provable guarantee on the approximation quality (albeit with strong assumptions). This also leads us to an important open direction from our work, i.e. gaining a better theoretical understanding of our online algorithms, potentially by proving approximation bounds going beyond worst-case analysis (Roughgarden, 2021). For instance, by assuming that the learned NDPP model satisfies natural assumptions like perturbation stability (Bilu & Linial, 2012; Makarychev et al., 2014; Angelidakis et al., 2017). For example, in the line of prior work (Lang et al., 2018; 2019; 2021a;b) studying MAP inference for Potts models. Another interesting direction is in providing parallelizable algorithms which use a small number of passes (greater than one but less than k) - similar to the k-means‖ (read as "k-means parallel") algorithm (Bahmani et al., 2012; Makarychev et al., 2020).

We have only studied 1-neighbor and 2-neighbor online local search algorithms in our paper. Extending them to arbitrary sizes of subsets (3-neighbor, etc.) is also another interesting open direction. Understanding at which point the degree of interactions cease to provide benefits that are worth the increase in memory/time constraints would be of interest to the DPP research community. Are pairwise interactions, as in 2-neighbor, sufficient to characterize most of the necessary NDPP properties?

Acknowledgments

We would like to thank all the ICML 2022 reviewers of our paper and also the ICLR 2022 reviewers who reviewed an earlier version of this work, for their very valuable feedback. Most of this work was done while AR was an intern with Adobe Research, San Jose, CA, USA in the summer of 2021. AR was also supported by NSF CCF-1652491 and NSF CCF-1955351 during the preparation of this manuscript.

References

Aho, A., Hopcroft, J., and Ullman, J. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

Anari, N. and Vuong, T.-D. From Sampling to Optimization on Discrete Domains with Applications to Determinant Maximization. In *Conference on Learning Theory (COLT)*, 2022.

Angelidakis, H., Makarychev, K., and Makarychev, Y. Algorithms for stable and perturbation-resilient problems. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 438–451, 2017.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

Bhaskara, A., Karbasi, A., Lattanzi, S., and Zadimoghaddam, M. Online MAP Inference of Determinantal Point Processes. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Bian, A. A., Buhmann, J. M., Krause, A., and Tschiatschek, S. Guarantees for Greedy Maximization of Non-submodular Functions with Applications. In *International Conference on Machine Learning (ICML)*, 2017.

Bilu, Y. and Linial, N. Are stable instances easy? *Combinatorics, Probability and Computing*, 21(5):643–660, 2012.

Borodin, A. Determinantal point processes. *The Oxford Handbook of Random Matrix Theory*, Oxford University Press, 2009.

Borodin, A. and El-Yaniv, R. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

Brunel, V.-E. Learning Signed Determinantal Point Processes through the Principal Minor Assignment Problem. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Brunel, V.-E., Moitra, A., Rigollet, P., and Urschel, J. Rates of estimation for determinantal point processes. In *Conference on Learning Theory (COLT)*, 2017.

Chen, D., Sain, S. L., and Guo, K. Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining. *Journal of Database Marketing & Customer Strategy Management*, 2012.

Chen, L., Zhang, G., and Zhou, H. Fast Greedy MAP Inference for Determinantal Point Process to Improve Recommendation Diversity. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Chen, W. and Ahmed, F. PaDGAN: A Generative Adversarial Network for Performance Augmented Diverse Designs. *Journal of Mechanical Design*, 143(3):031703, 2021.

Civril, A. and Magdon-Ismail, M. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science*, 410(47-49):4801–4811, 2009.

Derezinski, M. and Mahoney, M. W. Determinantal point processes in randomized numerical linear algebra. *Notices of the American Mathematical Society*, 68(1):34–45, 2021.

Fiedler, M. and Pták, V. Some generalizations of positive definiteness and monotonicity. Numerische Mathematik, 9(2):163–172, 1966.

Gartrell, M., Brunel, V.-E., Dohmatob, E., and Krichene, S. Learning Nonsymmetric Determinantal Point Processes. In Neural Information Processing Systems (NeurIPS), 2019.

Gartrell, M., Han, I., Dohmatob, E., Gillenwater, J., and Brunel, V.-E. Scalable Learning and MAP Inference for Nonsymmetric Determinantal Point Processes. In International Conference on Learning Representations (ICLR), 2021.

Gillenwater, J., Kulesza, A., and Taskar, B. Near-Optimal MAP Inference for Determinantal Point Processes. In Neural Information Processing Systems (NIPS), 2012.

Gillenwater, J., Kulesza, A., Mariet, Z., and Vassilvitskii, S. Maximizing Induced Cardinality Under a Determinantal Point Process. In Neural Information Processing Systems (NeurIPS), 2018.

Han, I. and Gillenwater, J. MAP Inference for Customized Determinantal Point Processes via Maximum Inner Product Search. In International Conference on Artificial Intelligence and Statistics (AISTATS), 2020.

Han, I., Kambadur, P., Park, K., and Shin, J. Faster Greedy MAP Inference for Determinantal Point Processes. In International Conference on Machine Learning, 2017.

Hough, J. B., Krishnapur, M., Peres, Y., and Virág, B. Determinantal Processes and Independence. Probability surveys, 3:206–229, 2006.

Indyk, P., Mahabadi, S., Oveis Gharan, S., and Rezaei, A. Composable Core-sets for Determinant Maximization: A Simple Near-Optimal Algorithm. In International Conference on Machine Learning (ICML), 2019.

Indyk, P., Mahabadi, S., Oveis Gharan, S., and Rezaei, A. Composable Core-sets for Determinant Maximization Problems via Spectral Spanners. In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2020.

Instacart. The Instacart Online Grocery Shopping Dataset, 2017. URL https://www.instacart.com/datasets/grocery-shopping-2017. Accessed August 2021.

Karp, R. M. On-Line Algorithms Versus Off-Line Algorithms: How Much is It Worth to Know the Future? In Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing, 1992.

Karp, R. M., Vazirani, U. V., and Vazirani, V. V. An Optimal Algorithm for On-Line Bipartite Matching. In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC), 1990.

Ko, C.-W., Lee, J., and Queyranne, M. An Exact Algorithm for Maximum Entropy Sampling. Operations Research, 1995.

Krause, A. and Golovin, D. Submodular Function Maximization. In Tractability: Practical Approaches to Hard Problems. Cambridge University Press, February 2014.

Kulesza, A. and Taskar, B. k-DPPs: Fixed-size Determinantal Point Processes. In International Conference on Machine Learning (ICML), 2011.

Kulesza, A. and Taskar, B. Determinantal Point Processes for Machine Learning. Foundations and Trends® in Machine Learning, 2012.

Lang, H., Sontag, D., and Vijayaraghavan, A. Optimality of Approximate Inference Algorithms on Stable Instances. In International Conference on Artificial Intelligence and Statistics, pp. 1157–1166, 2018.

Lang, H., Sontag, D., and Vijayaraghavan, A. Block Stability for MAP Inference. In The 22nd International Conference on Artificial Intelligence and Statistics, pp. 216–225, 2019.

Lang, H., Reddy, A., Sontag, D., and Vijayaraghavan, A. Beyond Perturbation Stability: LP Recovery Guarantees for MAP Inference on Noisy Stable Instances. In International Conference on Artificial Intelligence and Statistics, pp. 3043–3051. PMLR, 2021a.

Lang, H., Sontag, D., and Vijayaraghavan, A. Graph cuts always find a global optimum for Potts models (with a catch). In International Conference on Machine Learning, pp. 5990–5999. PMLR, 2021b.

Launay, C., Desolneux, A., and Galerne, B. Determinantal point processes for image processing. SIAM Journal on Imaging Sciences, 14(1):304–348, 2021.

Liu, P., Soni, A., Kang, E. Y., Wang, Y., and Parsana, M. Diversity on the Go! Streaming Determinantal Point Processes under a Maximum Induced Cardinality Objective. In Proceedings of the Web Conference (WWW), 2021.

Macchi, O. The Coincidence Approach to Stochastic Point Processes. Advances in Applied Probability, 7(1):83–122, 1975.

Mahabadi, S., Razenshteyn, I., Woodruff, D. P., and Zhou, S. Non-Adaptive Adaptive Sampling on Turnstile Streams. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC), 2020.

Makarychev, K., Makarychev, Y., and Vijayaraghavan, A. Bilu–Linial stable instances of max cut and minimum multiway cut. In Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms, pp. 890–906. SIAM, 2014.

Makarychev, K., Reddy, A., and Shan, L. Improved guarantees for k-means++ and k-means++ parallel. Advances in Neural Information Processing Systems, 33:16142–16152, 2020.

McFee, B., Bertin-Mahieux, T., Ellis, D. P., and Lanckriet, G. R. The Million Song Dataset Challenge. In Proceedings of the International Conference on World Wide Web (WWW), 2012.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. Lazier Than Lazy Greedy. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2015.

Muthukrishnan, S. Data streams: Algorithms and applications. Now Publishers Inc, 2005.

Nemhauser, G., Wolsey, L., and Fisher, M. An Analysis of Approximations for Maximizing Submodular Set Functions I. Mathematical Programming, 14(1), 1978.

Nguyen, V., Le, T., Yamada, M., and Osborne, M. A. Optimal Transport Kernels for Sequential and Parallel Neural Architecture Search. In International Conference on Machine Learning (ICML), 2021.

Perez-Beltrachini, L. and Lapata, M. Multi-Document Summarization with Determinantal Point Process Attention. Journal of Artificial Intelligence Research (JAIR), 71: 371–399, 2021.

Perez-Nieves, N., Yang, Y., Slumbers, O., Mguni, D. H., Wen, Y., and Wang, J. Modelling Behavioural Diversity for Learning in Open-Ended Games. In International Conference on Machine Learning (ICML), 2021.

Qian, X., Rossi, R. A., Du, F., Kim, S., Koh, E., Malik, S., Lee, T. Y., and Chan, J. Learning to Recommend Visualizations from Data. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 1359–1369, 2021.

Roughgarden, T. Beyond the Worst-Case Analysis of Algorithms. Cambridge University Press, 2021.

Sharma, M., Harper, F. M., and Karypis, G. Learning from Sets of Items in Recommender Systems. ACM Transactions on Interactive Intelligent Systems (TiiS), 9(4):1–26, 2019.

Shrivastava, A., Song, Z., and Xu, Z. Sublinear Least-Squares Value Iteration via Locality Sensitive Hashing. arXiv preprint arXiv:2105.08285, 2021.

Song, Z. and Yu, Z. Oblivious Sketching-based Central Path Method for Solving Linear Programming Problems. In 38th International Conference on Machine Learning (ICML), 2021.

Song, Z., Yang, S., and Zhang, R. Does Preprocessing Help Training Over-parameterized Neural Networks? Advances in Neural Information Processing Systems (NeurIPS), 34, 2021a.

Song, Z., Zhang, L., and Zhang, R. Training Multi-Layer Over-Parametrized Neural Network in Subquadratic Time. arXiv preprint arXiv:2112.07628, 2021b.

Xu, Z., Song, Z., and Shrivastava, A. Breaking the Linear Iteration Cost Barrier for Some Well-known Conditional Gradient Methods Using MaxIP Data-structures. Advances in Neural Information Processing Systems (NeurIPS), 34, 2021.

A. Streaming MAP Inference Details

Theorem 2. For a random-order arrival stream, if S is the solution output by Algorithm 1 at the end of the stream and $\sigma_{\min} > 1$ where $\sigma_{\min}$ and $\sigma_{\max}$ denote the smallest and largest singular values of L_S among all $S \subseteq [n]$ and $|S| \leq 2k$, then

$$\frac{E[\log\det(L_S)]}{\log(\mathrm{OPT})} \geq 1 - \frac{1}{\left(1-\frac{1}{e}\right)\left(2\log\sigma_{\max} - \log\sigma_{\min}\right)}$$

where $L_S = V_S^\top V_S + B_S^\top C B_S$ and $\mathrm{OPT} = \max_{R \subseteq [n];\, |R| = k} \det(V_R^\top V_R + B_R^\top C B_R)$.

Proof. As described in Algorithm 1, we will use $S_0, S_1, \ldots, S_k$ to denote the solution sets maintained by the algorithm, where S_i represents the solution set of size i. In particular, we have that $S_i = S_{i-1} \cup \{s_i\}$ where $s_i = \arg\max_{j \in B_i} f(S \cup \{j\})$ and B_i denotes the i'th partition i.e. $B_i := \{\frac{(i-1)n}{k}+1, \frac{(i-1)n}{k}+2, \ldots, \frac{in}{k}\}$.

For $i \in [k]$, let us use $X_i := [B_i \cap (S^* \setminus S_{i-1}) \neq \emptyset]$ to denote the event that there is at least one element of the optimal solution which has not already been picked by the algorithm in the batch B_i and $\ell_i := |S^* \setminus S_{i-1}|$. Then,

$$\begin{aligned}
\Pr[X_i] &= 1 - \Pr[X_i^c] \\
&= 1 - \left(1 - \frac{\ell_i}{n}\right)\left(1 - \frac{\ell_i}{n-1}\right)\cdots\left(1 - \frac{\ell_i}{n - \frac{n}{k} + 1}\right) \\
&\geq 1 - \left(1 - \frac{\ell_i}{n}\right)^{\frac{n}{k}} \\
&\geq 1 - e^{-\frac{\ell_i}{k}} \\
&\geq \frac{\ell_i}{k}\left(1 - \frac{1}{e}\right)
\end{aligned}$$

Here we use the facts: $e^x \geq 1 + x$ for all $x \in \mathbb{R}$, $1 - e^{-\frac{\ell}{k}}$ is concave as a function of ℓ, and $\ell \in [0, k]$.

For any element $s \in [n]$ and set $S \subseteq [n]$, let us use $f(s \mid S) := f(S \cup \{s\}) - f(S)$ to denote the marginal gain in f obtained by adding the element s to the set S. For any round $i \in [k]$, we then have that $f(S_i) - f(S_{i-1}) = f(s_i \mid S_{i-1})$.

Note that

$$E[f(s_i \mid S_{i-1}) \mid X_i] \geq \frac{\sum_{\omega \in \mathrm{OPT} \setminus S_{i-1}} f(\omega \mid S_{i-1})}{|\mathrm{OPT} \setminus S_{i-1}|}.$$

This happens due to the fact that conditioned on X_i, every element in $S^* \setminus S_{i-1}$ is equally likely to be present in B_i and the algorithm picks s_i such that $f(s_i \mid S_{i-1}) \geq f(s \mid S_{i-1})$ for all $s \in B_i$.

$$\begin{aligned}
E[f(s_i \mid S_{i-1}) \mid S_{i-1}] &= E[f(s_i \mid S_{i-1}) \mid S_{i-1}; X_i]\Pr[X_i] \\
&\quad + E[f(s_i \mid S_{i-1}) \mid S_{i-1}; X_i^c]\Pr[X_i^c] \\
&\geq E[f(s_i \mid S_{i-1}) \mid S_{i-1}; X_i]\Pr[X_i] \\
&\geq \frac{\ell_i}{k}\left(1 - \frac{1}{e}\right)\frac{\sum_{\omega \in S^* \setminus S_{i-1}} f(\omega \mid S_{i-1})}{|S^* \setminus S_{i-1}|} \\
&= \frac{\ell_i}{|S^* \setminus S_{i-1}|}\left(1 - \frac{1}{e}\right)\frac{1}{k}\sum_{\omega \in S^* \setminus S_{i-1}} f(\omega \mid S_{i-1}) \\
&= \left(1 - \frac{1}{e}\right)\frac{1}{k}\sum_{\omega \in S^* \setminus S_{i-1}} f(\omega \mid S_{i-1}) \\
&\geq \left(1 - \frac{1}{e}\right)\frac{1}{k}\left(f(S_{i-1} \cup S^*) - f(S_{i-1})\right) \\
&\geq \left(1 - \frac{1}{e}\right)\frac{1}{k}\left(\mathrm{OPT} - f(S_{i-1})\right)
\end{aligned}$$

For the last 2 inequalities, we use the fact that $f(S) = \log\det(L_S)$ is monotone non-decreasing and has a submodularity ratio of $\gamma = \left(2^{\frac{\log\lambda_{max}}{\log\lambda_{min}}} - 1\right)^{-1}$ when $\lambda_{min} > 1$ (Gartrell et al., 2021)[Eq. 45].

Taking expectation over all random draws of S_{i-1}, we get

$$E[f(s_i \mid S_{i-1})] \geq \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}(OPT - E[f(S_{i-1})])$$

Combining the above equation with $f(s_i \mid S_{i-1}) = f(S_i) - f(S_{i-1})$, we have

$$E[f(S_i)] - E[f(S_{i-1})] \geq \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}(OPT - E[f(S_{i-1})])$$

Next we have

$$-(OPT - E[f(S_i)]) + (OPT - E[f(S_{i-1})]) \geq \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}(OPT - E[f(S_{i-1})])$$

Re-organizing the above equation, we obtain

$$OPT - E[f(S_i)] \leq \left(1 - \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}\right)(OPT - E[f(S_{i-1})])$$

Applying the above equation recursively k times,

$$\begin{aligned} OPT - E[f(S_k)] &\leq \left(1 - \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}\right)^k (OPT - E[f(S_0)]) \\ &= \left(1 - \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}\right)^k OPT \end{aligned}$$

where the last step follows from $f(S_0) = 0$.

Re-organized the terms again, we have

$$\begin{aligned} E[f(S_k)] &\geq \left(1 - \left(1 - \left(1 - \frac{1}{e}\right)\frac{\gamma}{k}\right)^k\right) OPT \\ &\geq \left(1 - e^{-(1-\frac{1}{e})\gamma}\right) OPT \end{aligned}$$

When we substitute $\gamma = \left(2^{\frac{\log\lambda_{max}}{\log\lambda_{min}}} - 1\right)^{-1}$, we get our final inequality:

$$E[f(S_k)] \geq \left(1 - \frac{1}{\lambda_{min}^{(1-\frac{1}{e})/(2\log\lambda_{max} - \log\lambda_{min})}}\right) OPT$$

B. Hard instance for One-Pass MAP Inference of NDPPs

Outline: We will now give a high-level description of a hard instance for online MAP inference of NDPPs (this is inspired by (Anari & Vuong, 2022, Example 5)) . Suppose we have a total of $2d$ items consisting of d pairs of complementary items like modem-router, printer-ink cartridge, pencil-eraser etc. Let us use $\{1, 1^c, 2, 2^c, \ldots, d, d^c\}$ to denote them. Any item i is independent of every other item other than it's complement i^c. Individually, $\Pr[\{i\}] = \Pr[\{i^c\}] = 0$. And $\Pr[\{i, i^c\}] = x_i^2$ with $x_i > 0$ for all $i \in [d]$. Also, we have $\Pr[\{i, j\}] = 0$ for any $i \neq j$. Suppose any of our online algorithms are given the sequence $\{1, 2, 3, \ldots, d, r^c\}$ where $r \in [d]$ is some arbitrary item and the algorithm needs to pick 2 items i.e. $k = 2$. Then, $\mathrm{OPT} > 0$ whereas all of our online algorithms (Online LSS, Online 2-neighbor, Online-Greedy) will fail to output a valid solution.

Details: Let $0 < x_1 < x_2 < \cdots < x_d$. Suppose

$$C = \begin{bmatrix} \begin{pmatrix} 0 & x_1 \\ -x_1 & 0 \end{pmatrix} & & & \\ & \begin{pmatrix} 0 & x_2 \\ -x_2 & 0 \end{pmatrix} & & \\ & & \ddots & \\ & & & \begin{pmatrix} 0 & x_d \\ -x_d & 0 \end{pmatrix} \end{bmatrix}$$

$C \in \mathbb{R}^{2d \times 2d}$ is a skew-symmetric (i.e. $C = -C^\top$) block diagonal matrix where the blocks are of the form $\begin{pmatrix} 0 & x_i \\ -x_i & 0 \end{pmatrix}$.

Suppose we have a total of $2d$ items consisting of d pairs of complementary items. We use $\{1, 1^c, 2, 2^c, \ldots, d, d^c\}$ to denote them. Let $v_i = v_{i^c} = 0 \; \forall i \in [d]$ and $b_1 = e_1; b_{1^c} = e_2; \ldots; b_{d^c} = e_{2d}$ where $e_1; e_2; \ldots; e_{2d}$ are the standard unit vectors in $\mathbb{R}^{2d}$ i.e. $B = I_{2d}$.

For a pair of complementary items $S = \{i, i^c\}$, $f(S) = x_i^2$. Without loss of generality, consider $S = \{1, 1^c\}$. Then we can compute $B_S^\top C B_S$ as follows:

$$\begin{aligned} B_S^\top C B_S &= \begin{pmatrix} e_1 & e_2 \end{pmatrix}^\top C \begin{pmatrix} e_1 & e_2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & x_1 & 0 & 0 & \cdots & 0 \\ -x_1 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} e_1 & e_2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & x_1 \\ -x_1 & 0 \end{pmatrix} \end{aligned}$$

In this case, we have $f(S) = x_1^2$.

For any pair of non-complementary items $S = \{i_1, i_2\}$ where the indices are distinct, $f(S) = 0$. Without loss of generality, we can consider $S = \{1, 2\}$. Then,

$$\begin{aligned} B_S^\top C B_S &= \begin{pmatrix} e_1 & e_3 \end{pmatrix}^\top C \begin{pmatrix} e_1 & e_3 \end{pmatrix} \\ &= \begin{pmatrix} 0 & x_1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & x_2 & \cdots & 0 \end{pmatrix} \begin{pmatrix} e_1 & e_3 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{aligned}$$

And so, we have that $f(S) = 0$.

C. Experiments and Datasets details

All experiments were performed using a standard desktop computer (Quad-Core Intel Core i7, 16 GB RAM) using many real-world datasets composed of sets (or baskets) of items from some ground set of items:

- UK Retail: This is an online retail dataset consisting of sets of items all purchased together by users (in a single transaction) (Chen et al., 2012). There are 19,762 transactions (sets of items purchased together) that consist of 3,941 items. Transactions with more than 100 items are discarded.
- MovieLens: This dataset contains sets of movies that users watched (Sharma et al., 2019). There are 29,516 sets consisting of 12,549 movies.
- Amazon Apparel: This dataset consists of 14,970 registries (sets) from the apparel category of the Amazon Baby Registries dataset, which is a public dataset that has been used in prior work on NDPPs (Gartrell et al., 2021; 2019). These apparel registries are drawn from 100 items in the apparel category.
- Amazon 3-category: We also use a dataset composed of the apparel, diaper, and feeding categories from Amazon Baby Registries, which are the most popular categories, giving us 31,218 registries made up of 300 items (Gartrell et al., 2019).
- Instacart: This dataset represents sets of items purchased by users on Instacart (Instacart, 2017). Sets with more than 100 items are ignored. This gives 3.2 million total item-sets from 49,677 unique items.
- Million Song: This is a dataset of song playlists put together by users where every playlist is a set (basket) of songs played by Echo Nest users (McFee et al., 2012). Playlists with more than 150 songs are discarded. This gives 968,674 playlists from 371,410 songs.
- Customer Dashboards: This dataset consists of dashboards or baskets of visualizations created by users (Qian et al., 2021). Each dashboard represents a set of visualizations selected by a user. There are 63436 dashboards (baskets/sets) consisting of 1206 visualizations.
- Web Logs: This dataset consists of sets of webpages (baskets) that were all visited in the same session. There are 2.8 million baskets (sets of webpages) drawn from 2 million webpages.
- Company Retail: This dataset contains the set of items viewed (or purchased) by a user in a given session. Sets (baskets) with more than 100 items are discarded. This results in 2.5 million baskets consisting of 107349 items.

The last two datasets are proprietary Adobe data. The learning algorithm of (Gartrell et al., 2021) takes as input a parameter d, which is the embedding size for V, B, C. We used $d = 10$ for all datasets other than Instacart, Customer Dashboards, Company Retail where $d = 50$ is used and Million Song, where $d = 100$ is used. For all of our results in 7, we set $k = 8$ and choose $= 1{:}1$.

D. Additional Experimental Results

D.1. Number of Determinant Computations

We perform several experiments comparing the number of determinant computations (as a system-independent proxy for time) of all our online algorithms. We do not compare with of ine greedy here because that algorithm doesn't explicitly compute all the determinants. Results comparing the number of determinant computations as a function of the number of data points analyzed for a variety of datasets are provided in Figure 2. Online-2-neighbor requires the most number of determinant computations but also gives the best results in terms of solution value. Online-LSS and Online-Greedy use very similar number of determinant computations.

Figure 2. Results comparing the number of determinant computations as a function of the number of data points analyzed for all our online algorithms. Online-2-neighbor requires the most number of determinant computations but also gives the best results in terms of solution value. Online-LSS and Online-Greedy use very similar number of determinant computations.

D.2. Number of Swaps

Results comparing the number of swaps (as a measure of solution consistency) of all our online algorithms can be found in Figure 3. Online-Greedy has the most number of swaps and therefore the least consistent solution set. On most datasets, the number of swaps by Online-2-neighbor is very similar to Online-LSS.

Figure 3. Results comparing the number of swaps of all our online algorithms. Online-Greedy does the most number of swaps and therefore has the least consistent solution set. On most datasets, the number of swaps by Online-2-neighbor is very similar to Online-LSS.

D.3. Random Streams

We also investigate our algorithms under the random stream paradigm. For this setting, we use some of the previous real-world datasets, and randomly permute the order in which the data appears in the stream. We do this 100 times and report the average of solution values in Figure 4 and the average of number of determinant computations and swaps in Figure 5. We observe that Online-2-neighbor and Online-LSS give very similar performance in this regime and they are always better than Online-Greedy.

D.4. Ablation study varying

To study the effect of in Online-LSS (Algorithm 2), we vary $\in \{0.05, 0.1, 0.3, 0.5\}$ and analyze the value of the obtained solutions, number of determinant computations, and number of swaps. We notice that, in general, the solution quality, number of determinant computations, and the number of swaps increase as decreases. Results are provided in Figure 6.

Figure 4. Solution quality as a function of the number of data points analyzed in the random stream paradigm. Online-2-neighbor and Online-LSS give very similar performance in this setting and they are always better than Online-Greedy.

Figure 5. Number of determinant computations and swaps as a function of the number of data points analyzed in the random stream setting. Online-2-Neighbor needs more determinant computations than Online-LSS but has very similar number of swaps in this setting. Note that = 1

.

Figure 6. Performance of Online-LSS varying for $k = 8$. Solution quality, number of determinant computations, and number of swaps seem to increase with decreasing .